

A brick wall on the left side of a blue background. The bricks are reddish-brown with white mortar. The wall is on the left side of the frame, and the blue background is on the right side.

Building Java Programs

Chapters 3-4:
Using Objects

Chapter outline

- using objects
- text processing
 - char type

A brick wall is visible on the left side of the slide, extending from the bottom to the top. The bricks are reddish-brown with white mortar. The background is a solid blue color.

Using objects

reading: 3.3

Objects

- So far, we have seen:
 - **methods**, which represent behavior
 - **variables**, which represent data (categorized by **types**)
- It is possible to create new types that are combinations of the existing types.
 - Such types are called *object types* or *reference types*.
 - Languages such as Java in which you can do this are called *object-oriented* programming languages.
- We will learn how to use some of Java's objects.
 - In Chapter 8 we will learn to create our own types of objects.

Objects and classes

- **object:** An entity that contains data and behavior.
 - There are variables inside the object, representing its data.
 - There are methods inside the object, representing its behavior.
- **class:** A program, or a template for a type of objects.
- **Examples:**
 - The class `String` represents objects that store text characters.
 - The class `Point` represents objects that store (x, y) data.
 - The class `Scanner` represents objects that read information from the keyboard, files, and other sources.

Constructing objects

- **construct:** To create a new object.
 - Objects are constructed with the `new` keyword.
 - Most objects must be constructed before they can be used.
- Constructing objects, general syntax:
<type> <name> = new <type> (<parameters>);
 - Examples:

```
Point p = new Point(7, -4);
DrawingPanel window = new DrawingPanel(300, 200);
Color orange = new Color(255, 128, 0);
```
 - Classes' names are usually uppercase (e.g. `Point`, `Color`).
 - Strings are also objects, but are constructed without `new` :

```
String name = "Amanda Ann Camp";
```

Calling methods of objects

- Objects contain methods that can be called by your program.
 - For example, a `String`'s methods manipulate or process its text.
 - When we call an object's method, we are sending a message to it.
 - We specify which object we are talking to, then the method's name.
- Calling a method of an object, general syntax:
`<variable> . <method name> (<parameters>)`

- The results will be different from one object to another.

- Examples:

```
String gangsta = "G., Ali";  
System.out.println(gangsta.length());           // 7  
  
Point p1 = new Point(3, 4);  
Point p2 = new Point(0, 0);  
System.out.println(p1.distance(p2));           // 5.0
```

Point objects

- Java has a class of objects named `Point`.
 - To use `Point`, you must write: `import java.awt.*;`

- Constructing a `Point` object, general syntax:

```
Point <name> = new Point(<x>, <y>);
```

```
Point <name> = new Point(); // the origin, (0, 0)
```

- Examples:

```
Point p1 = new Point(5, -2);
```

```
Point p2 = new Point();
```

- `Point` objects are useful for several reasons:
 - They store two values, an (x, y) pair, in a single variable.
 - They have useful methods we can call in our programs.

Point data and methods

- Data stored in each `Point` object:

Field name	Description
<code>x</code>	the point's x-coordinate
<code>y</code>	the point's y-coordinate

- Methods of each `Point` object:

Method name	Description
<code>distance(<i>p</i>)</code>	how far away the point is from point <i>p</i>
<code>setLocation(<i>X</i>, <i>y</i>)</code>	sets the point's x and y to the given values
<code>translate(<i>dx</i>, <i>dy</i>)</code>	adjusts the point's x and y by the given amounts

- Point objects can also be printed using `println` statements:

```
Point p = new Point(5, -2);  
System.out.println(p);    // java.awt.Point[x=5,y=-2]
```

Using Point objects

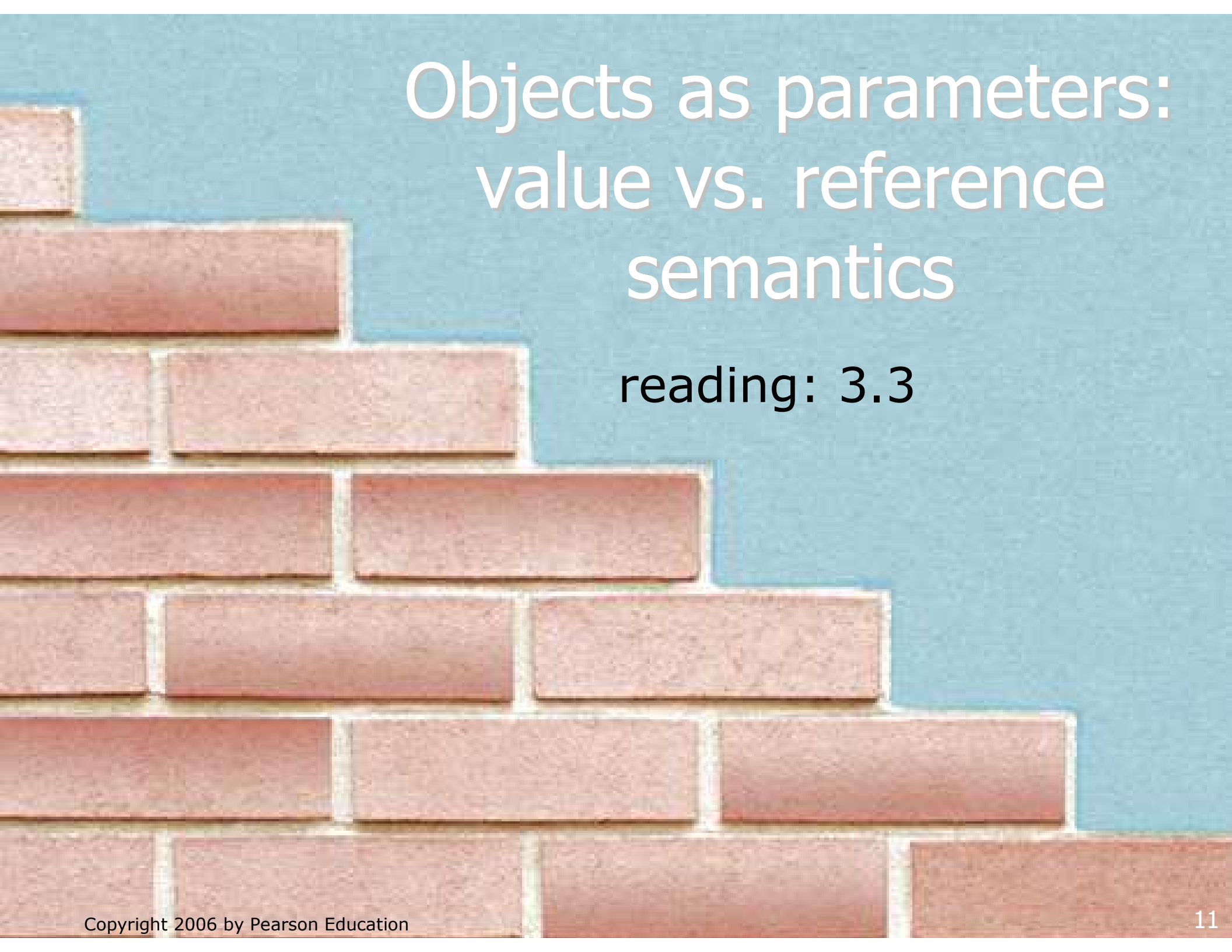
- An example program that uses `Point` objects:

```
import java.awt.*;

public class PointMain {
    public static void main(String[] args) {
        // construct two Point objects
        Point p1 = new Point(7, 2);
        Point p2 = new Point(4, 3);

        // print each point and their distance apart
        System.out.println("p1 is " + p1);
        System.out.println("p2: (" + p2.x + ", " + p2.y + ")");
        System.out.println("distance = " + p1.distance(p2));

        // translate the point to a new location
        p2.translate(1, 7);
        System.out.println("p2: (" + p2.x + ", " + p2.y + ")");
        System.out.println("distance = " + p1.distance(p2));
    }
}
```

A brick wall on the left side of a blue background. The bricks are reddish-brown with white mortar. The wall is partially visible, extending from the left edge towards the center of the frame.

Objects as parameters: value vs. reference semantics

reading: 3.3

Swapping primitive values

- Consider the following code to swap two `int` variables:

```
public static void main(String[] args) {  
    int a = 7;  
    int b = 35;  
    System.out.println(a + " " + b);  
  
    // swap a with b  
    a = b;  
    b = a;  
  
    System.out.println(a + " " + b);  
}
```

- What is wrong with this code? What is its output?

Swapping, corrected

- When swapping, you should set aside one variable's value into a temporary variable, so it won't be lost.
 - Better code to swap two `int` variables:

```
public static void main(String[] args) {  
    int a = 7;  
    int b = 35;  
    System.out.println(a + " " + b);  
  
    // swap a with b  
    int temp = a;  
    a = b;  
    b = temp;  
  
    System.out.println(a + " " + b);  
}
```

A swap method?

- Swapping is a common operation, so we might want to make it into a method.
 - Does the following `swap` method work? Why or why not?

```
public static void main(String[] args) {  
    int a = 7;  
    int b = 35;  
    System.out.println(a + " " + b);  
    // swap a with b  
    swap(a, b);  
    System.out.println(a + " " + b);  
}
```

```
public static void swap(int a, int b) {  
    int temp = a;  
    a = b;  
    b = temp;  
}
```

Value semantics

- **value semantics:** Behavior where variables are copied when assigned to each other or passed as parameters.
 - Primitive types in Java use value semantics.
 - When one variable is assigned to another, the value is copied.
 - Modifying the value of one variable does not affect others.

- **Example:**

```
int x = 5;
```

```
int y = x;           // x = 5, y = 5
```

```
y = 17;              // x = 5, y = 17
```

```
x = 8;               // x = 8, y = 17
```



Modifying primitive parameters

- When we call a method and pass primitive variables' values as parameters, we can assign new values to the parameters inside the method.
 - But this does not affect the value of the variable that was passed; its value was copied, and the two variables are otherwise distinct.
 - Example:

```
public static void main(String[] args) {  
    int x = 1;      x 1  
    foo(x);  
    System.out.println(x);    // output: 1  
}
```

value 1 is copied into parameter

```
public static void foo(int x) {  
    x = 2;  
}
```

x ~~1~~ 2

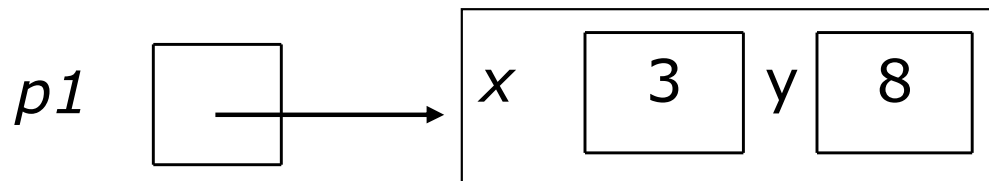
parameter's value is changed to 2
(variable x in main is unaffected)

Reference semantics

- **reference semantics:** Behavior where variables refer to a common value when assigned to each other or passed as parameters.
 - Objects in Java use reference semantics.
 - Object variables do not actually store an object; they store the address of an object's location in the computer's memory.
 - Variables for objects are called *reference variables*.
 - We often draw reference variables as small boxes that point an arrow toward the object they refer to.

- **Example:**

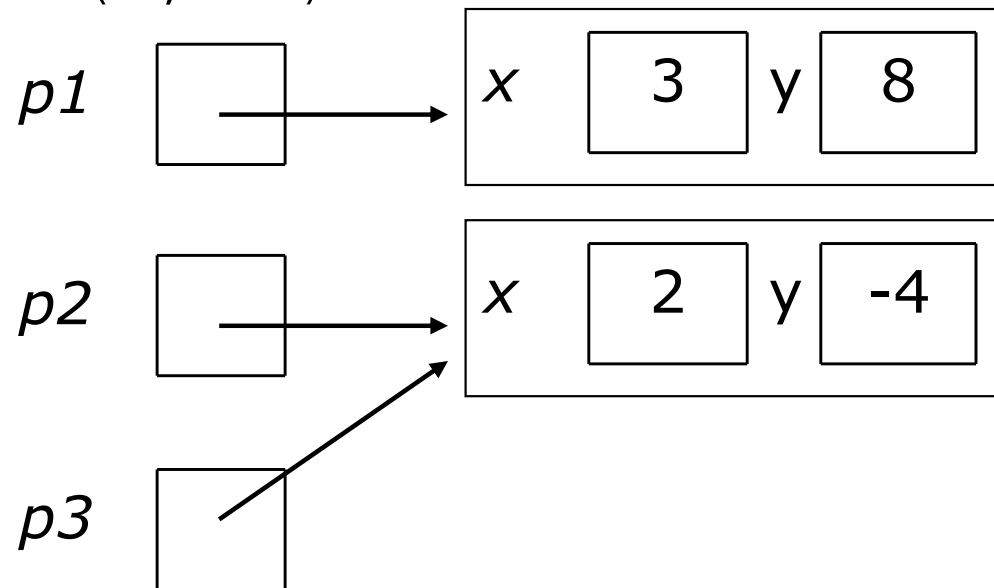
```
Point p1 = new Point(3, 8);
```



Multiple references

- If two reference variables are assigned to refer to the same object, the object is *not* copied.
 - Both variables literally share the same object.
 - Calling a method on either variable will modify the same object.

```
Point p1 = new Point(3, 8);  
Point p2 = new Point(2, -4);  
Point p3 = p2;
```



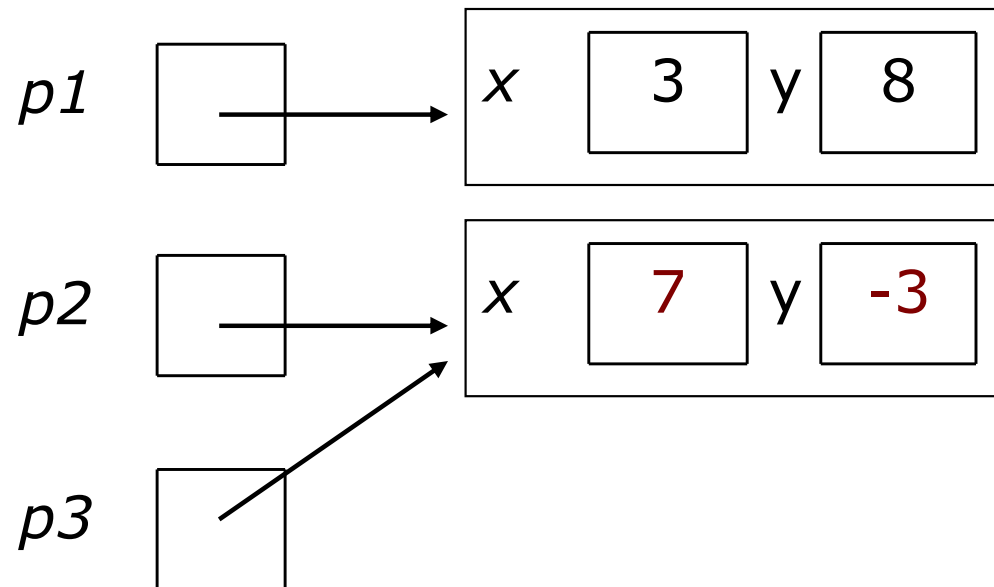
- Here 3 variables refer to 2 objects. If we change *p3*, will *p2* change? If we change *p2*, will *p3* change?

Multiple references

- If two variables refer to the same object, modifying one of them *will* also make a change in the other:

```
p3.translate(5, 1);
```

```
System.out.println("(" + p2.x + " " + p2.y + ")");
```



OUTPUT:

```
(7, -3)
```

Why references?

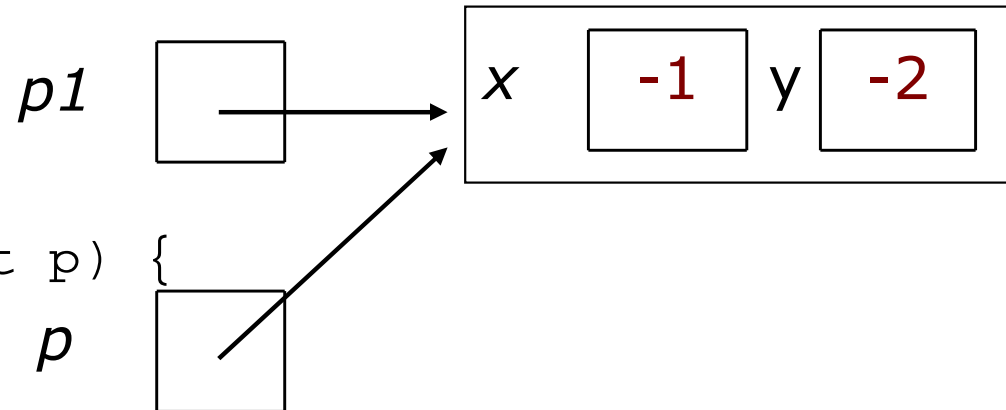
- The fact that objects are passed by reference was done for several reasons:
 - *efficiency*. Objects can be large, bulky things. Having to copy them every time they are passed as parameters would slow down the program.
 - *sharing*. Since objects hold important state and have behavior to modify that state, it is often desirable for them to be shared by parts of the program when they're passed as parameters. We want any changes to occur to the same object.

Objects as parameters

- When an object is passed as a parameter, the object is *not* copied. The same object is shared by the original variable and parameter.
 - If a method is called on the parameter, it *will* affect the original object that was passed to the method.
 - Since the variable `p1` and the parameter `p` refer to the same object, modifying one *will* also make a change in the other:

```
public static void main(String[] args) {  
    Point p1 = new Point(2, 3);  
    example(p1);  
}
```

```
public static void example(Point p) {  
    p.setLocation(-1, -2);  
}
```



String objects

- **string**: A sequence of text characters.
 - One of the most common types of objects.
 - In Java, strings are represented as objects of class `String`.
- `String` variables can be declared and assigned, just like primitive values:

```
String <name> = "<text>";
```

```
String <name> = <expression that produces a String>;
```

- Unlike most other objects, a `String` is not created with `new`.
- Examples:

```
String name = "Marla Singer";
```

```
int x = 3, y = 5;
```

```
String point = "(" + x + ", " + y + ")";
```

Indexes

- The characters in a `String` are each internally numbered with an *index*, starting with 0:

- Example:

```
String name = "P. Diddy";
```

index	0	1	2	3	4	5	6	7
character	'P'	'.'	' '	'D'	'i'	'd'	'd'	'y'

- Individual characters are represented inside the `String` by values of a primitive type called `char`.
 - Literal `char` values are surrounded with apostrophe (single-quote) marks, such as `'a'` or `'4'`.
 - An escape sequence can be represented as a `char`, such as `'\n'` (new-line character) or `'\''` (apostrophe).

String methods

- Useful methods of each `String` object:

Method name	Description
<code>charAt(<i>index</i>)</code>	character at a specific index
<code>indexOf(<i>str</i>)</code>	index where the start of the given string appears in this string (-1 if it is not there)
<code>length()</code>	number of characters in this string
<code>substring(<i>index1</i>, <i>index2</i>)</code>	the characters in this string from <i>index1</i> (inclusive) to <i>index2</i> (<u>exclusive</u>)
<code>toLowerCase()</code>	a new string with all lowercase letters
<code>toUpperCase()</code>	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String example = "speak friend and enter";
```

```
System.out.println(example.toUpperCase());
```


String method examples

```
//      index 012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";
System.out.println(s1.length());           // 12
System.out.println(s1.indexOf("e"));       // 8
System.out.println(s1.substring(1, 4));    // tua

String s3 = s2.toUpperCase();
System.out.println(s3.substring(6, 10));   // STEP

String s4 = s1.substring(0, 6);
System.out.println(s4.toLowerCase());     // stuart
```

Modifying Strings

- The methods that appear to modify a string (substring, toLowerCase, toUpperCase, etc.) actually create and return a new string.

```
String s = "lil bow wow";  
s.toUpperCase();  
System.out.println(s);    // output: lil bow wow
```

- If you want to modify the variable, you must reassign it to store the result of the method call:

```
String s = "lil bow wow";  
s = s.toUpperCase();  
System.out.println(s);    // output: LIL BOW WOW
```

String methods

- Given the following string:

```
String book = "Building Java Programs";
```

- How would you extract the word "Java" ?
- How would you change `book` to store:
"BUILDING JAVA PROGRAMS" ?
- How would you extract the first word from any general string?

Method name	Description
<code>charAt(<i>index</i>)</code>	character at a specific index
<code>indexOf(<i>str</i>)</code>	index where the start of the given string appears in this string (-1 if it is not there)
<code>length()</code>	number of characters in this string
<code>substring(<i>index1</i>, <i>index2</i>)</code>	the characters in this string from <i>index1</i> (inclusive) to <i>index2</i> (exclusive)
<code>toLowerCase()</code>	a new string with all lowercase letters
<code>toUpperCase()</code>	a new string with all uppercase letters

A brick wall on the left side of a blue background. The bricks are reddish-brown with white mortar lines. The wall is partially visible, extending from the left edge towards the center of the frame.

Text processing with String and char

reading: 4.3 - 4.4

Comparing objects

- Relational operators such as `<` and `==` only behave correctly on primitive values.
 - The `==` operator on `Strings` often evaluates to `false` even when the `Strings` have the same letters in them.

- Example (*incorrect*):

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- This example code will compile, but it will never print the message, even if the user does type `Barney`

The equals method

- Objects (such as `String`, `Point`, and `Color`) should be compared for equality by calling a method named `equals`.

- Example (correct):

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

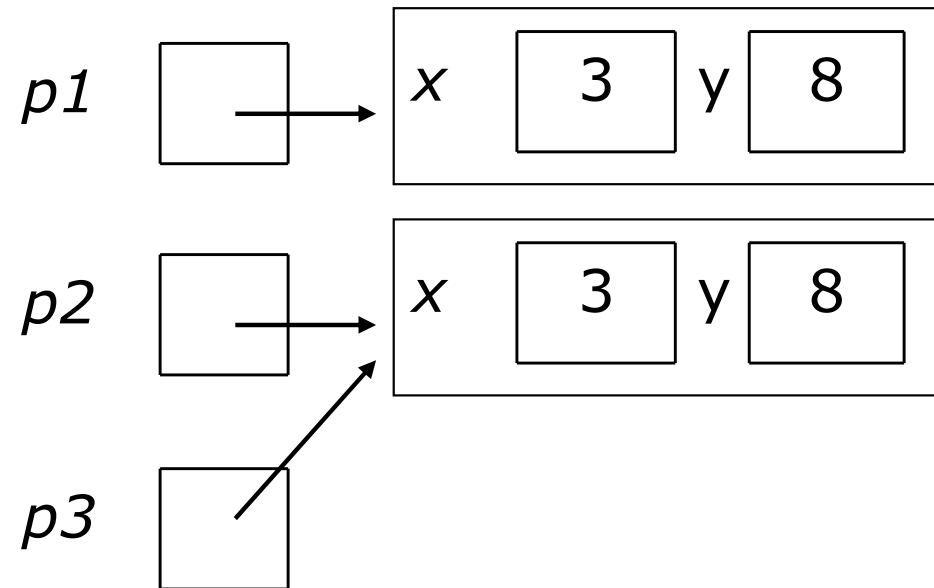
Another example

- The `==` operator on objects actually compares whether two variables refer to the same object.
- The `equals` method compares whether two objects have the same state as each other.
 - Given the following code:

```
Point p1 = new Point(3, 8);  
Point p2 = new Point(3, 8);  
Point p3 = p2;
```

- What is printed?

```
if (p1 == p2) {  
    System.out.println("1");  
}  
if (p1.equals(p2)) {  
    System.out.println("2");  
}  
if (p2 == p3) {  
    System.out.println("3");  
}
```



String condition methods

- There are several methods of a `String` object that can be used as conditions in `if` statements:

Method	Description
<code>equals(str)</code>	whether two strings contain exactly the same characters
<code>equalsIgnoreCase(str)</code>	whether two strings contain the same characters, ignoring upper vs. lower case differences
<code>startsWith(str)</code>	whether one string contains the other's characters at its start
<code>endsWith(str)</code>	whether one string contains the other's characters at its end

String condition examples

- Hypothetical examples, assuming the existence of various `String` variables:

- ```
if (title.endsWith("Ph. D. ")) {
 System.out.println("How's life in the ivory tower?");
}
```
- ```
if (fullName.startsWith("Queen")) {  
    System.out.println("Greetings, your majesty.");  
}
```
- ```
if (lastName.equalsIgnoreCase("lumberg")) {
 System.out.println("I need your TPS reports!");
}
```
- ```
if (name.toLowerCase().indexOf("jr.") >= 0) {  
    System.out.println("You share your parent's name.");  
}
```

Type char

- **char**: A primitive type representing single characters.
 - Individual characters inside a `String` are stored as `char` values.
 - Literal `char` values are surrounded with apostrophe (single-quote) marks, such as `'a'` or `'4'` or `'\n'` or `'\''`
 - It is legal to have variables, parameters, returns of type `char`

```
char letter = 'S';  
System.out.println(letter);           // S
```

The charAt method

- The characters of a string can be accessed as `char` values using the `String` object's `charAt` method.

```
String word = console.next();
char firstLetter = word.charAt(0);
if (firstLetter == 'c') {
    System.out.println("That's good enough for me!");
}
```

- We often use `for` loops that print or examine each character.

```
String name = "tall";
for (int i = 0; i < name.length(); i++) {
    System.out.println(title.charAt(i));
}
```

Output:

```
t
a
l
l
```

Text processing

- **text processing:** Examining, editing, formatting text.
 - Text processing often involves `for` loops that examine the characters of a string one by one.
 - You can use `charAt` to search for or count occurrences of a particular value in a string.

```
// Returns the count of occurrences of c in s.
public static int count(String s, char c) {
    int count = 0;
    for (int i = 0; i < s.length(); i++) {
        if (s.charAt(i) == 't') {
            count++;
        }
    }
    return count;
}
```

- `count("mississippi", 'i')` returns 4

Other things to do with char

- char values can be concatenated with strings.

```
char initial = 'P';  
System.out.println(initial + " Diddy");
```

- You can compare char values with relational operators:

- 'a' < 'b' and 'Q' != 'q'
- Note that you cannot use these operators on a String.

- An example that prints the alphabet:

```
for (char c = 'a'; c <= 'z'; c++) {  
    System.out.print(c);  
}
```

Type casting

- **type cast:** A conversion from one type to another.
 - Common uses:
 - To promote an `int` into a `double` to achieve exact division.
 - To truncate a `double` from a real number to an integer.
- type cast syntax:

(*<type>*) *<expression>*

Examples:

- `double result = (double) 19 / 5; // 3.8`
- `int result2 = (int) result; // 3`

More about type casting

- Type casting has high precedence and only casts the item immediately next to it.
 - `double x = (double) 1 + 1 / 2; // 1`
 - `double y = 1 + (double) 1 / 2; // 1.5`
- You can use parentheses to force evaluation order.
 - `double average = (double) (a + b + c) / 3;`
- A conversion to `double` can be achieved in other ways.
 - `double average = 1.0 * (a + b + c) / 3;`

char/int and type casting

- All `char` values are assigned numbers internally by the computer, called *ASCII* values.
 - Examples:
'A' is 65, 'B' is 66, 'a' is 97, 'b' is 98
 - Mixing `char` and `int` causes automatic conversion to `int`.
'a' + 10 is 107, 'A' + 'A' is 130
 - To convert an integer into the equivalent character, type cast it.
(char) ('a' + 2) is 'c'

char vs. String

- 'h' is a char

```
char c = 'h';
```

- char values are primitive; you cannot call methods on them
- can't say `c.length()` or `c.toUpperCase()`

- "h" is a String

```
String s = "h";
```

- Strings are objects; they contain methods that can be called
- *can* say `s.length()` \longrightarrow 1
- *can* say `s.toUpperCase()` \longrightarrow "H"
- *can* say `s.charAt(0)` \longrightarrow 'h'

- What is `s + 1` ? What is `c + 1` ?

- What is `s + s` ? What is `c + c` ?

Text processing questions

- Write a method named `pigLatinWord` that accepts a `String` as a parameter and outputs that word in simplified Pig Latin, by placing the word's first letter at the end followed by the suffix *ay*.
 - `pigLatinWord("hello")` prints `ello-hay`
 - `pigLatinWord("goodbye")` prints `oodbye-gay`
- Write methods named `encode` and `decode` that accept a `String` as a parameter and outputs that `String` with each of its letters increased or decreased by 1.
 - `encode("hello")` prints `ifmmp`
 - `decode("ifmmp")` prints `hello`

Text processing question

- Write a method `printName` that accepts a full name as a parameter, and prints the last name followed by a comma, followed by the first name and middle initial.
 - For example,
`printName("James Tiberius Kirk");` would output:
Kirk, James T.

Method name	Description
<code>charAt(<i>index</i>)</code>	character at a specific index
<code>indexOf(<i>str</i>)</code>	index where the start of the given string appears in this string (-1 if it is not there)
<code>length()</code>	number of characters in this string
<code>substring(<i>index1</i>, <i>index2</i>)</code>	the characters in this string from <i>index1</i> (inclusive) to <i>index2</i> (exclusive)
<code>toLowerCase()</code>	a new string with all lowercase letters
<code>toUpperCase()</code>	a new string with all uppercase letters